Article
Joel Solon · Oct 30, 2019 4m read

Collecting Performance Data While Running Unit Tests

A few years ago, I was teaching the basics of our %UnitTest framework during Caché Foundations class (now called Developing Using InterSystems Objects and SQL). A student asked if it was possible to collect performance statistics while running unit tests. A few weeks later, I added some additional code to the %UnitTest examples to answer this question. I ' m finally sharing it on the Community.

The %SYSTEM,Process class provides several metrics that you can collect for a process (other than Duration).

- Duration
- Lines Executed
- Global References
- System CPU Time
- User CPU Time
- Disk Read Time

To enable **any** unit test to collect these stats, create a subclass of %UnitTest.TestCase, and add properties.

```
Class Performance.TestCase Extends %UnitTest.TestCase
{
Property Duration As %Time;
Property Lines As %Integer;
Property Globals As %Integer;
Property SystemCPUTime As %Integer;
Property UserCPUTime As %Integer;
Property DiskReadTime As %Integer;
}
```

Any specific unit test class you create should inherit from your new subclass instead of %UnitTest.TestCase.

In the subclass, use OnBeforeOneTest() to initialize the stats collection for each unit test. For everything except DiskReadTime, the code initializes the property with the current value.

```
/// initialize performance stats
Method OnBeforeOneTest(testname As %String) As %Status
{
   // initialize with current values
   set ..Duration = $zh
   set ..Lines = $system.Process.LinesExecuted()
   set ..Globals = $system.Process.GlobalReferences()
   set ..SystemCPUTime = $piece(CPUTime, ",", 1)
   set ..UserCPUTime = $piece(CPUTime, ",", 2)
   // reset disk read time to 0 and start counting
   do $system.Process.ResetDiskReadTiming()
   do $system.Process.EnableDiskReadTiming()
   return $$$OK
}
```

Use OnAfterOneTest() to finalize the stats collection for each unit test. For everything except DiskReadTime, the code subtracts the initial value from the current value.

```
 /// Finalize performance stats
/// This is where you could also add code to save the counters to a separate table for analysis.
Method OnAfterOneTest(testname As %String) As %Status
{
    set ..Duration = $zh - ..Duration
    set ..Lines = $system.Process.LinesExecuted() - ..Lines
    set ..Globals = $system.Process.GlobalReferences() - ..Globals
     set CPUTime = $system.Process.GetCPUTime()
    set ..SystemCPUTime = $piece(CPUTime, ",", 1) - ..SystemCPUTime
    set ..UserCPUTime = $piece(CPUTime, ",", 2) - ..UserCPUTime
    // get disk read time and stop counting
    set ..DiskReadTime = $system.Process.DiskReadMilliseconds()
    do $system.Process.DisableDiskReadTiming()
    // add message to unit test log
    set msg = "Performance: " _    "Duration: " _..Duration _
   ", Lines: " _..Lines _
   ", Globals: " _..Globals _
   ", System CPU Time: " _(..SystemCPUTime / 1000) _
   ", User CPU Time: " _(..UserCPUTime / 1000) _
   ", Disk Read Time: " _(..DiskReadTime / 1000)
    do $$$LogMessage(msg)
    return $$$OK
}
```

There ' s one more little trick. You may want to run your unit test**with or without** collecting statistics. So, the code where you are invoking your unit tests must take an argument (could be a %Boolean 1 or 0) and somehow pass that in. The methods that actually run the tests (such as RunTest() or one of the other Run*() methods) take an array as the 3rd argument, passed by reference. Here ' s an example snippet:

```
 // create an array to hold the logging argument (1 or 0) and pass it by reference
set p("logging") = logging
do ##class(%UnitTest.Manager).RunTest(test, qualifiers, .p)
```

The value you pass in the array can be accessed in OnBeforeOneTest() and OnAfterOneTest(). Add this as the first line in both methods:

```
if (..Manager.UserFields.GetAt("logging") = 0) { return $$$OK }
```

That ' s it! Looking forward to your questions, comments, and additional ideas.

#Best Practices #Code Snippet #Performance #Testing #Caché #InterSystems IRIS

Source URL:https://community.intersystems.com/post/collecting-performance-data-while-running-unit-tests